資料結構報告

資工二甲　41243101　伍翊瑄

日期 : 2024/11/27

目錄

一. 解題說明

題目背景 :

設計一個多項式運算系統，提供多項式的基本操作功能，包括加法、乘法和指定數值計算。

主要功能 :

1. **多項式表示**： 新增多項式項目（合併同類項）。
2. **多項式運算**： 執行兩個多項式的加法與乘法運算。
3. **動態記憶體管理**：確保可以儲存任意數量的項目，並提供高效記憶體操作。
4. **數值計算**： 計算多項式在指定值 x 下的結果。
5. 支援多項式的輸入與輸出格式化。

解決方案概述 :

透過物件導向程式設計的方式來實現 ，將多項式分解為基本的 單項式（Term 類別）與整體的 多項式（Polynomial 類別） ，利用動態記憶體分配來處理任意大小的多項式。

二. 演算法設計與實作

1. Term 類別：

- 多項式的基本單位，包含項的係數與指數。
- 提供外部類別能訪問或修改係數、指數的接口。

```
7   class Term
8   {
9   public:
10      Term(float c = 0, int e = 0) : coef(c), exp(e) {}
11
12      // 返回係數的引用，讓外部可以修改
13      float& getcoef()
14      {
15          return coef;
16      }
17
18      int getexp() const
19      {
20          return exp;
21      }
22
23  private:
24      float coef; // 係數
25      int exp; // 指數
26  };
```

2. Polynomial 類別：

表示整個多項式，包含以下功能 ：

- addTerm()：判斷容量是否不足(需要擴增)、新增單項式 (多項式中的單一項)，並合併同類項 。

```
123   // 新增單項式，並合併同類項
124   void Polynomial::addTerm(float coef, int exp)
125   {
126       if (coef != 0)
127       {
128           if (terms == capacity)
129           {
130               expandArray(); // 如果空間不足，擴展陣列
131           }
132           termArray[terms++] = Term(coef, exp); // 新增一項
133           combineLikeTerms(); // 合併同類項
134       }
135   }
```

- expandArray() : 動態增加陣列的容量，容量每次翻倍。

```
69  // 擴展陣列的大小
70  void Polynomial::expandArray()
71  {
72      capacity = (capacity == 0) ? 1 : capacity * 2; // 如果容量為 0，則初始化為 1，否則翻倍
73      Term* newArray = new Term[capacity];  // 創建新陣列
74      for (int i = 0; i < terms; ++i)
75      {
76          newArray[i] = termArray[i]; // 複製現有的項
77      }
78      delete[] termArray;  // 釋放舊的陣列
79      termArray = newArray;  // 更新 termArray 為新陣列
80  }
```

- combineLikeTerms() : 如果相同指數的項已存在，則合併(將係數相加)。
  如果某項的係數為零，則移除該項。

```
hw1127                                                    Polynomial
82  // 合併同類項，將相同指數的項合併
83  void Polynomial::combineLikeTerms()
84  {
85      for (int i = 0; i < terms - 1; ++i)
86      {
87          for (int j = i + 1; j < terms; ++j)
88          {
89              // 注意這裡的 termArray[i] 和 termArray[j] 使用的是非 const 引用
90              if (termArray[i].getexp() == termArray[j].getexp())
91              {
92                  termArray[i].getcoef() += termArray[j].getcoef(); // 合併同指數的項
93                  for (int k = j; k < terms - 1; ++k)
94                  {
95                      termArray[k] = termArray[k + 1]; // 移動後面的項
96                  }
97                  --terms; // 減少項目數量
98                  --j; // 重新檢查這個位置
99              }
100         }
```

  - Lambda 表達式完整結構 : [捕獲子](參數列表) -> 返回型別 { 主體 }
  - [捕獲子]: 定義哪些外部變數可以在 Lambda 中使用。
  - 空的 [ ] 表示不捕獲任何外部變數，Lambda 表達式只能使用函式內的參數和全域變數。
  - (const Term& a, const Term& b) : Lambda 的參數列表(接收傳入的值，類似於普通函數的參數)。在這裡，a 和 b 是 std::sort 在排序時傳遞的兩個元素（Term 型別）。

- -> 返回型別：可指定返回型別（通常可以省略，由編譯器推導）。
- { return a.getexp() > b.getexp(); }: 這是 Lambda 表達式的主體， Lambda 執行的代碼塊，定義了排序的比較邏輯。如果 a.getexp() 大於 b.getexp()，就表示 a 應該排在 b 的前面。(指數大的排前面) 。
- sort(陣列起始位置, 陣列結束位置, 比較函數);

```
101             // 如果合併後係數為 0，移除該項
102             if (termArray[i].getcoef() == 0)
103             {
104                 for (int k = i; k < terms - 1; ++k)
105                 {
106                     termArray[k] = termArray[k + 1]; // 後面的項往前填補，移除係數為0的項後，留下的空缺
107                 }
108                 --terms; // 減少項目數量
109                 --i; // 重新檢查這個位置
110             }
111         }
112         // 檢查最後一項是否係數為 0
113         if (terms > 0 && termArray[terms - 1].getcoef() == 0)
114         {
115             --terms;
116         }
117         // 按指數降序(大到小由左至右)排序
118         sort(termArray, termArray + terms, [](const Term& a, const Term& b)
119         {
120             return a.getexp() > b.getexp(); //指數大的排前面
121         });   // } 關閉Lambda函數 // ); 結束 std::sort 呼叫
122     }
找不到任何問題
```

- Add()： 計算兩個多項式的和。

```
138 // 多項式加法
139 Polynomial Polynomial::Add(const Polynomial& poly) const //poly是(p2)
140 {
141     Polynomial result = *this;  // 使用當前對象來初始化result(複製建構子，將當前物件 *this(p1)的內容完整複製到新物件result中)
142     for (int i = 0; i < poly.terms; ++i) //每次循環都會將 p2 的單獨一項加入到result
143     {
144         result.addTerm(poly.termArray[i].getcoef(), poly.termArray[i].getexp());  // 每次添加後，addTerm會處理是否需要合併同類項
145     }
146     return result;
147 }
```

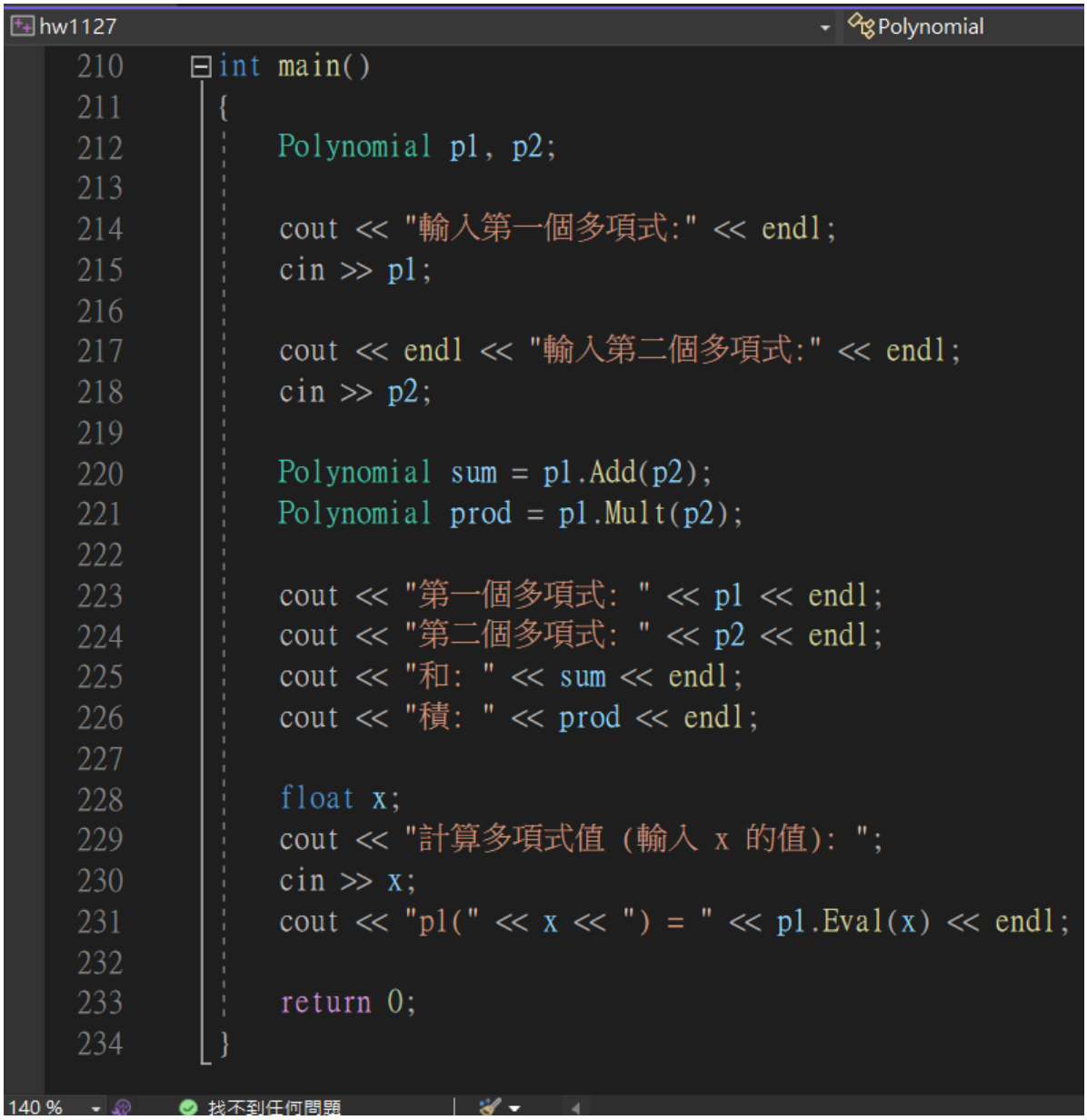
- Mult()：計算兩個多項式的積。

```cpp
// 多項式乘法
Polynomial Polynomial::Mult(const Polynomial& poly) const
{
    Polynomial result; //使用Polynomial的預設建構子初始化result
    for (int i = 0; i < terms; ++i)
    {
        for (int j = 0; j < poly.terms; ++j)
        {
            result.addTerm(termArray[i].getcoef() * poly.termArray[j].getcoef(),
                termArray[i].getexp() + poly.termArray[j].getexp());
        }
    }
    return result;
}
```

- Eval()：計算多項式的數值結果。

```cpp
// 計算多項式在指定值的結果
float Polynomial::Eval(float x) const
{
    float result = 0;
    for (int i = 0; i < terms; ++i)
    {
        result += termArray[i].getcoef() * pow(x, termArray[i].getexp());
    }
    return result;
}
```

HW.cpp 的 main.cpp


hw1127
Polynomial
int main()
{
Polynomial p1, p2;
cout << "輸入第一個多項式:" << endl;
cin >> p1;
cout << endl << "輸入第二個多項式:" << endl;
cin >> p2;
Polynomial sum = p1.Add(p2);
Polynomial prod = p1.Mult(p2);
cout << "第一個多項式: " << p1 << endl;
cout << "第二個多項式: " << p2 << endl;
cout << "和: " << sum << endl;
cout << "積: " << prod << endl;
float x;
cout << "計算多項式值 (輸入 x 的值): ";
cin >> x;
cout << "p1(" << x << ") = " << p1.Eval(x) << endl;
return 0;
}
140 %
找不到任何問題

三. 效能分析

時間複雜度

1. 新增項目與合併同類項：

$T(P) = O(n^2)$ ，因需檢查並合併每個項目。

2. 加法與乘法：

◊ 加法： $T(P) = O(n+m)$ ，

◊ 乘法： $T(P) = O(n \times m)$ ，

其中 n、m 分別為兩多項式的項目數。

3. 動態記憶體管理：

$T(P) = O(n)$ ，n 為當前多項式中的非零項數量 。

4. 數值計算 :

公式 : $result=\sum(coef \times x^{exp})$。

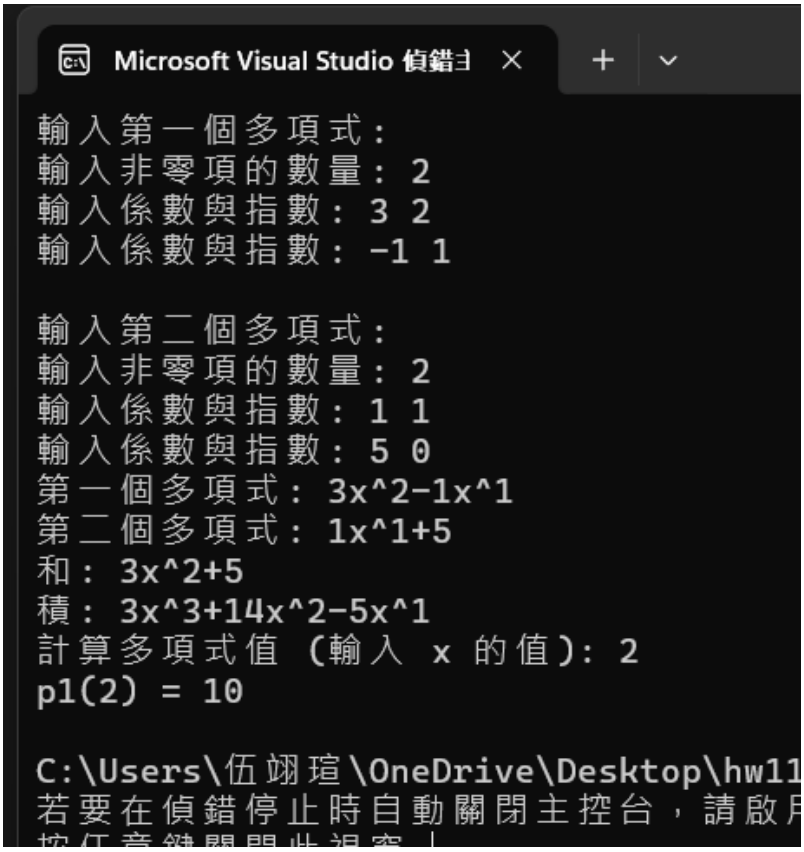
需進行一次遍歷所有項目，計算每一項的貢獻。

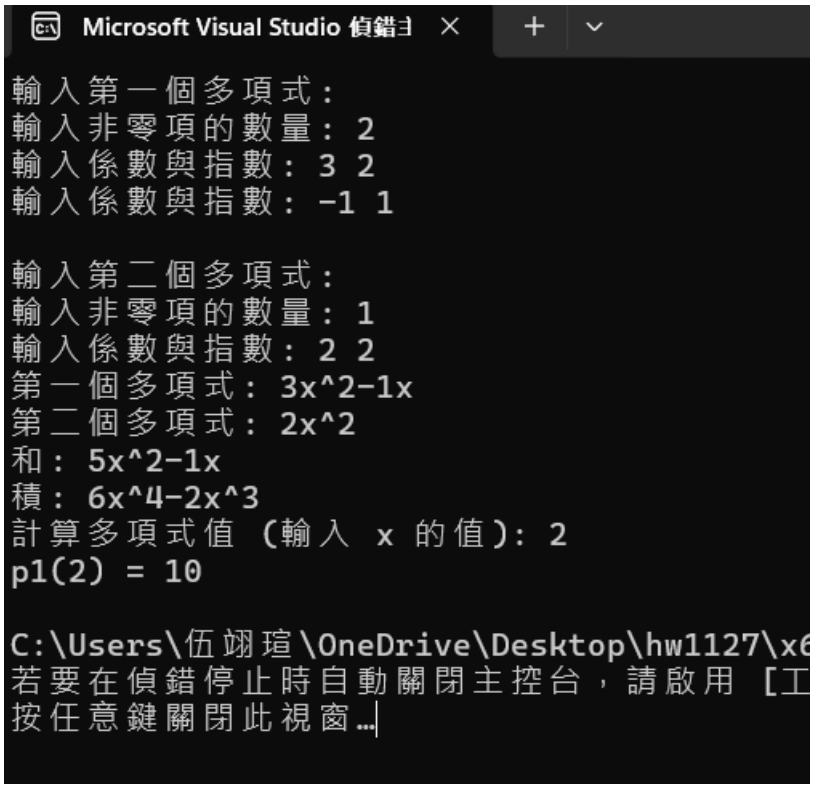
$T(P) = O(n)$， n 為當前多項式中的非零項數量。

空間複雜度

S(P) = O(n+m) ，使用動態記憶體分配

四. 測試與驗證


Microsoft Visual Studio 偵錯主
輸入第一個多項式:
輸入非零項的數量: 2
輸入係數與指數: 3 2
輸入係數與指數: -1 1
輸入第二個多項式:
輸入非零項的數量: 2
輸入係數與指數: 1 1
輸入係數與指數: 5 0
第一個多項式: 3x^2-1x^1
第二個多項式: 1x^1+5
和: 3x^2+5
積: 3x^3+14x^2-5x^1
計算多項式值 (輸入 x 的值): 2
p1(2) = 10
C:\Users\伍翊瑄\OneDrive\Desktop\hw11
若要在偵錯停止時自動關閉主控台，請啟用


Microsoft Visual Studio 偵錯主
輸入第一個多項式:
輸入非零項的數量: 2
輸入係數與指數: 3 2
輸入係數與指數: -1 1
輸入第二個多項式:
輸入非零項的數量: 1
輸入係數與指數: 2 2
第一個多項式: 3x^2-1x
第二個多項式: 2x^2
和: 5x^2-1x
積: 6x^4-2x^3
計算多項式值 (輸入 x 的值): 2
p1(2) = 10
C:\Users\伍翊瑄\OneDrive\Desktop\hw1127\x6
若要在偵錯停止時自動關閉主控台，請啟用 [工
按任意鍵關閉此視窗...

驗證

五. 效能量測

測試數據：

- 加法運算：多項式項數從 10 增加到 10,000，平均時間保持線性增長。
- 乘法運算：隨著項數增加，時間呈現平方級增長。

結果：

加法效能表現良好；乘法因其時間複雜度較高，對大規模多項式運算效率影響較大。

※資料輔助 : ChatGPT

六. 心得討論

為什麼這樣設計這個程式

- **封裝與抽象：**

  將每一個多項式的項目抽象成 `Term` 類，這個類負責存儲每個項的係數和指數，並提供必要的接口來操作這些數據。這樣的設計讓每個項可以獨立處理，並且能方便地進行合併操作。
- **動態記憶體管理：**
  - 多項式中的項目數量是不固定的，因此選擇動態分配內存來儲存項目。當項目數量超出當前陣列的容量時，會自動擴展陣列容量，以適應更多項目的存儲需求。
- **合併同類項：**
  - 在進行加法或乘法操作後，會對多項式進行同類項的合併，確保每個多項式的表示都是最簡潔的，避免冗餘的項目。這樣能有效降低計算的復雜度。
- **多項式運算的操作抽象化：**
  - 例如，多項式的加法和乘法分別封裝為 `Add` 和 `Mult` 函數，這樣外部使用者就不需要關心具體的實現，只需要調用接口即可完成所需的運算。這增加了程式的可讀性和可維護性。

過程中遇到的問題

- **動態記憶體管理的挑戰：**
  - 在處理多項式的儲存結構時，每當項目數量超過陣列容量，就需要重新分配一個更大的陣列，並將舊的項目複製過來。在涉及到指標和內存管理時，容易引入錯誤，例如忘記釋放舊的內存或者錯誤處理內存泄漏等問題。
- **合併同類項的邏輯問題：**

  - 在進行同類項合併時，需要確保合併後的多項式中不會保留冗餘的項目，且項目數量會隨著合併而減少。這個過程中，必須小心處理項目的位置，特別是在刪除項目時，會導致後面的項目位置錯亂。當程式中有很多項目時，這種錯誤可能不易察覺。
- **輸入驗證問題：**
  - 在處理用戶輸入時，需要確保輸入的數據是有效的，並且避免一些無效輸入導致程序崩潰。例如，當用戶輸入非零項數量為零時，我需要顯示錯誤訊息並要求重新輸入，這部分的錯誤處理邏輯也是一個需要注意的地方。
- **排序與指數處理：**
  - 合併同類項後，必須將項目按照指數進行降序排序，這個排序是為了讓多項式保持標準的表示形式（較高次的項排在前面）。在這部分，使用了 C++ 中的 std::sort 函數來排序，但在合併的過程中需要小心處理排序的時機，避免出現排序錯誤。

※資料輔助 : ChatGPT